Tampereen yliopisto
Tampere University

Lecture 05

Cloud orchestration, DevOps, CD

Kari Systä

22.09.2020

Content

- Course practicalities
- Next exercise – orchestration
- Recap and continuation of some DevOps stuff
  - How about embedded?
  - Our related research

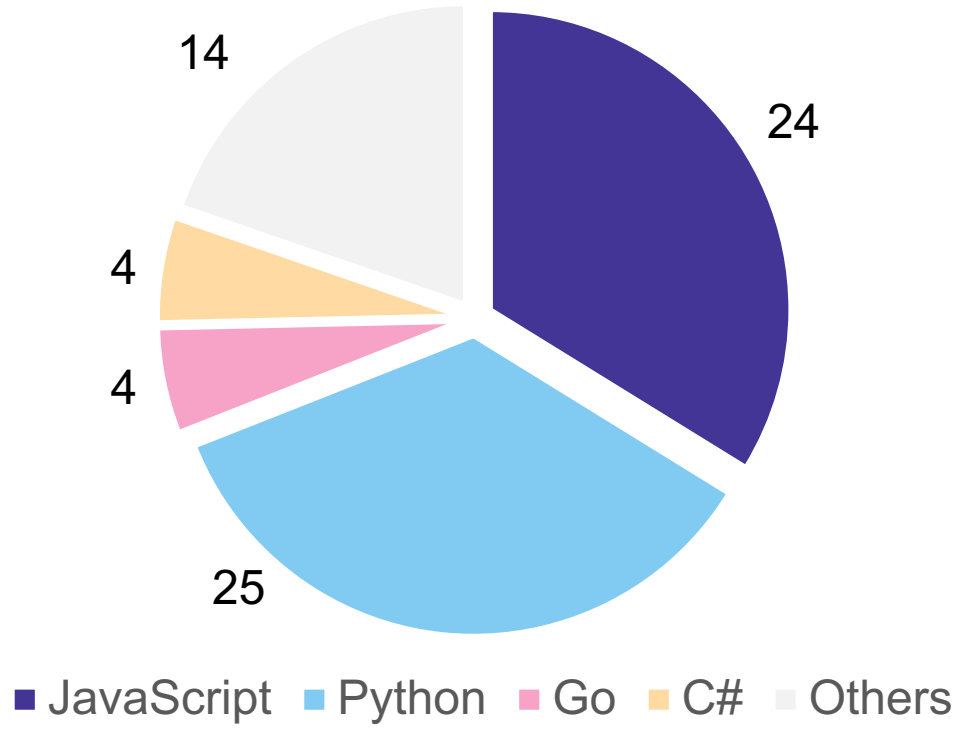
Docker exercise

- 70 students returned
- Delay “penalty” 0.2p / day => maximum 1.2p (out of 7)
- In general good responses

Programming languages

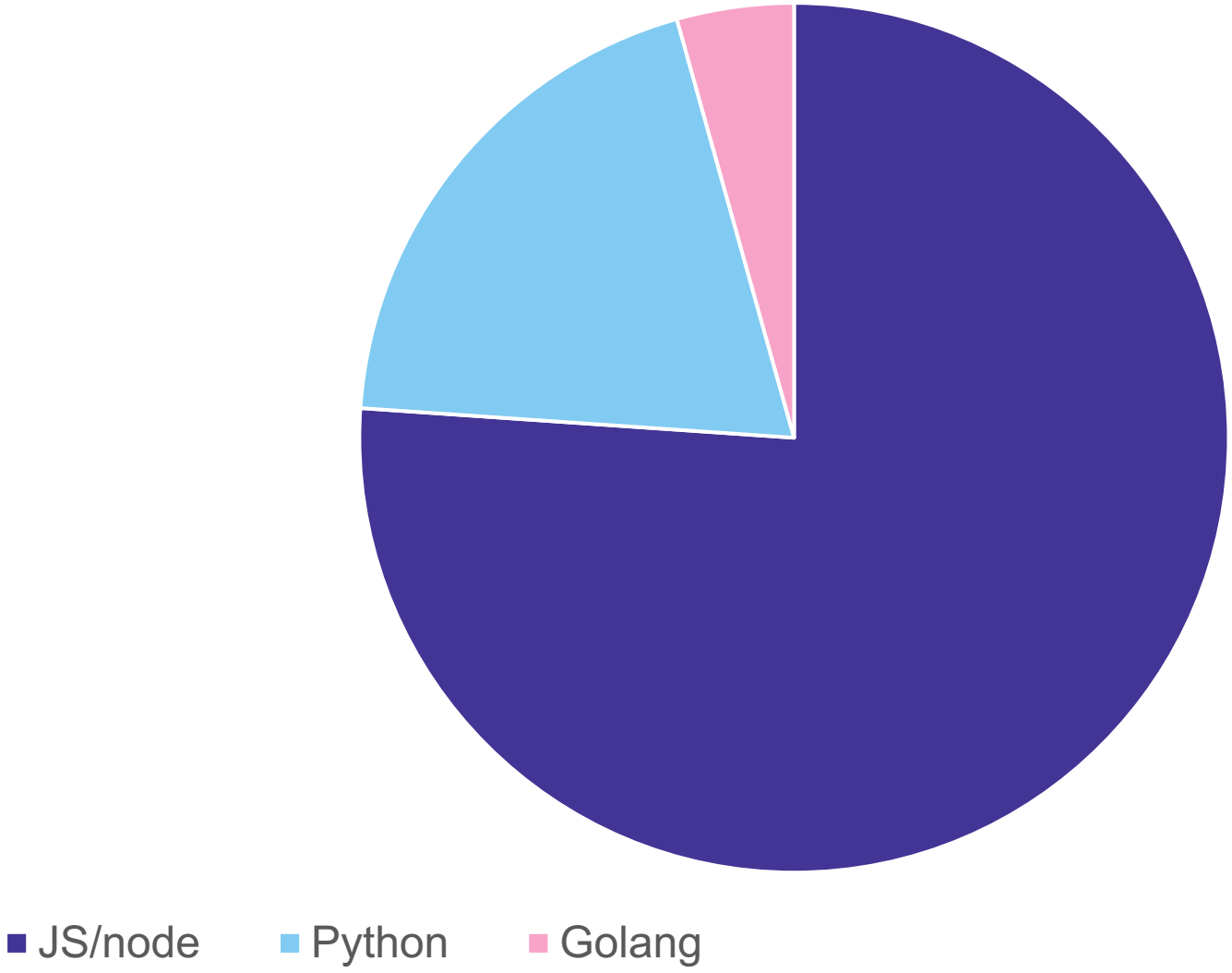
Others:

C, C++
PHP Rust
Java Julia
Haskel AWK
Racket Shell
HTML



Tampereen yliopisto
Tampere University

Last year (compose exercise)



Tampereen yliopisto
Tampere University

Next exercise: orchestration in practice

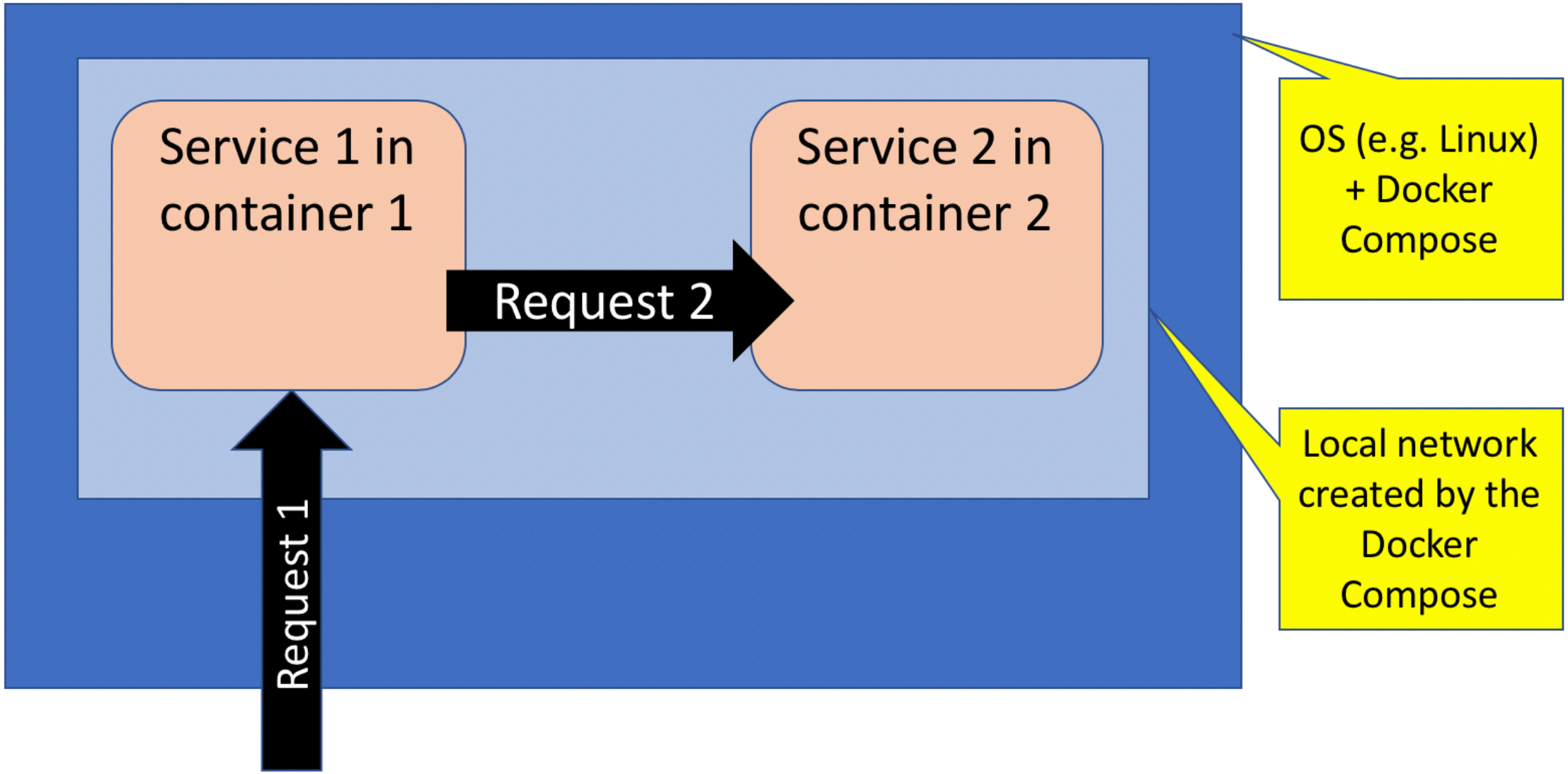
- The task is use docker_compose to ”orchestrate” a system of two services.
- The services are simple (naïve), but you need to implement them
  - You can use your favourite programming language and platform
- You have time until October 1st

Tampereen yliopisto
Tampere University

What is ”cloud orchestration”?

Two results of googling

- **Orchestration** is the automated configuration, coordination, and management of computer systems and software
- Cloud orchestration is the use of programming technology to manage the interconnections and interactions among workloads on public and private cloud infrastructure. It connects automated tasks into a cohesive workflow to accomplish a goal, with permissions oversight and policy enforcement.


Service 1 in container 1
Request 2
Service 2 in container 2
Request 1
OS (e.g. Linux) + Docker Compose
Local network created by the Docker Compose

Docker compose

```
version: '3'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
      - logvolume01:/var/log
    links:
      - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

```
$ docker-compose up –d
$ ./run_tests
$ docker-compose down
```

Remember pets and cattle?

```yaml
version: "3.7"
services:
  redis:
    image: redis:alpine
    deploy:
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 120s
```

Tampereen yliopisto
Tampere University

YAML

- Wikipedia: **YAML** ("YAML Ain't Markup Language") is a human-readable data-serialization language. It is commonly used for configuration files
- Spaces for indentation – have a syntactical meaning
- https://www.tutorialspoint.com/yaml/yaml_basics.htm

YAML -> JSON

```yaml
version: '3'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
      - logvolume01:/var/log
    links:
      - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

```json
{
  "version": "3",
  "services": {
    "web": {
      "build": ".",
      "ports": [
        "5000:5000"
      ],
      "volumes": [
        ".:/code",
        "logvolume01:/var/log"
      ],
      "links": [
        "redis"
      ]
    },
    "redis": {
      "image": "redis"
    }
  },
  "volumes": {
    "logvolume01": {}
  }
}
```

Tampereen yliopisto
Tampere University

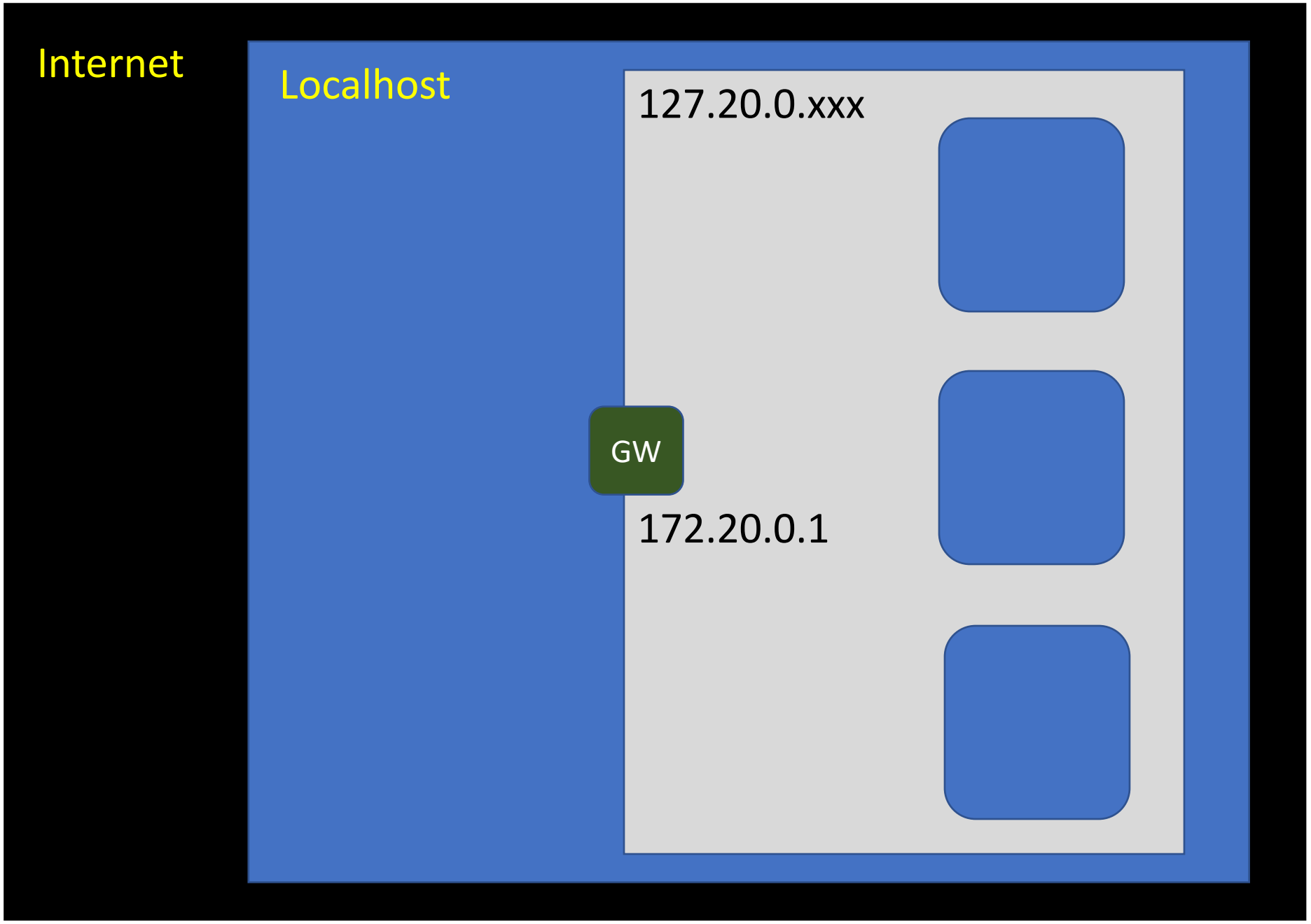
Nice looking tutorial

- https://www.baeldung.com/docker-compose

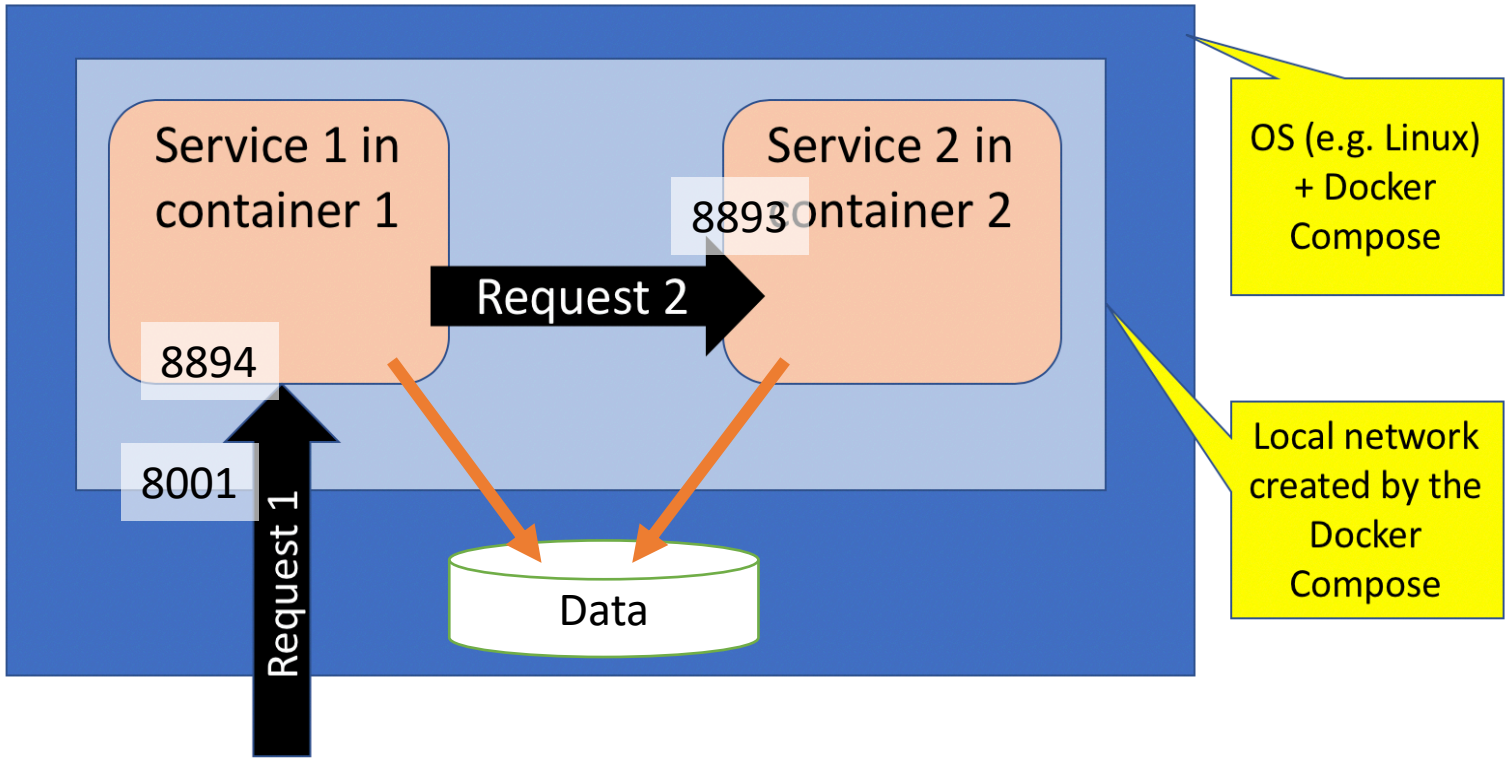
Networking aspects

```
version: '3'
services:
  pinger:
    image: "pinger"
    ports:
      - "8893:8893"
    networks:
      - pingnet
    volumes:
      - ./data:/data
    environment:
      ServiceName: service_2
  pingrelay:
    build: "pingrelay"
    ports:
      - "8004:8894"
    networks:
      - pingnet
    volumes:
      - ./data:/data
    environment:
      ServiceName: service_1
networks:
   pingnet:

volumes:
  data: {}
```


```
[
  {
    "Name": "composetest_pingnet",
    "Id": "42d79573d3b3cf…",
    "Created": "2019-02-14T20:08:36.226402086+02:00",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.20.0.0/16",
          "Gateway": "172.20.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": true,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {},
    "Options": {},
    "Labels": {
      "com.docker.compose.network": "pingnet",
      "com.docker.compose.project": "composetest",
      "com.docker.compose.version": "1.23.1"
    }
  }
]
```

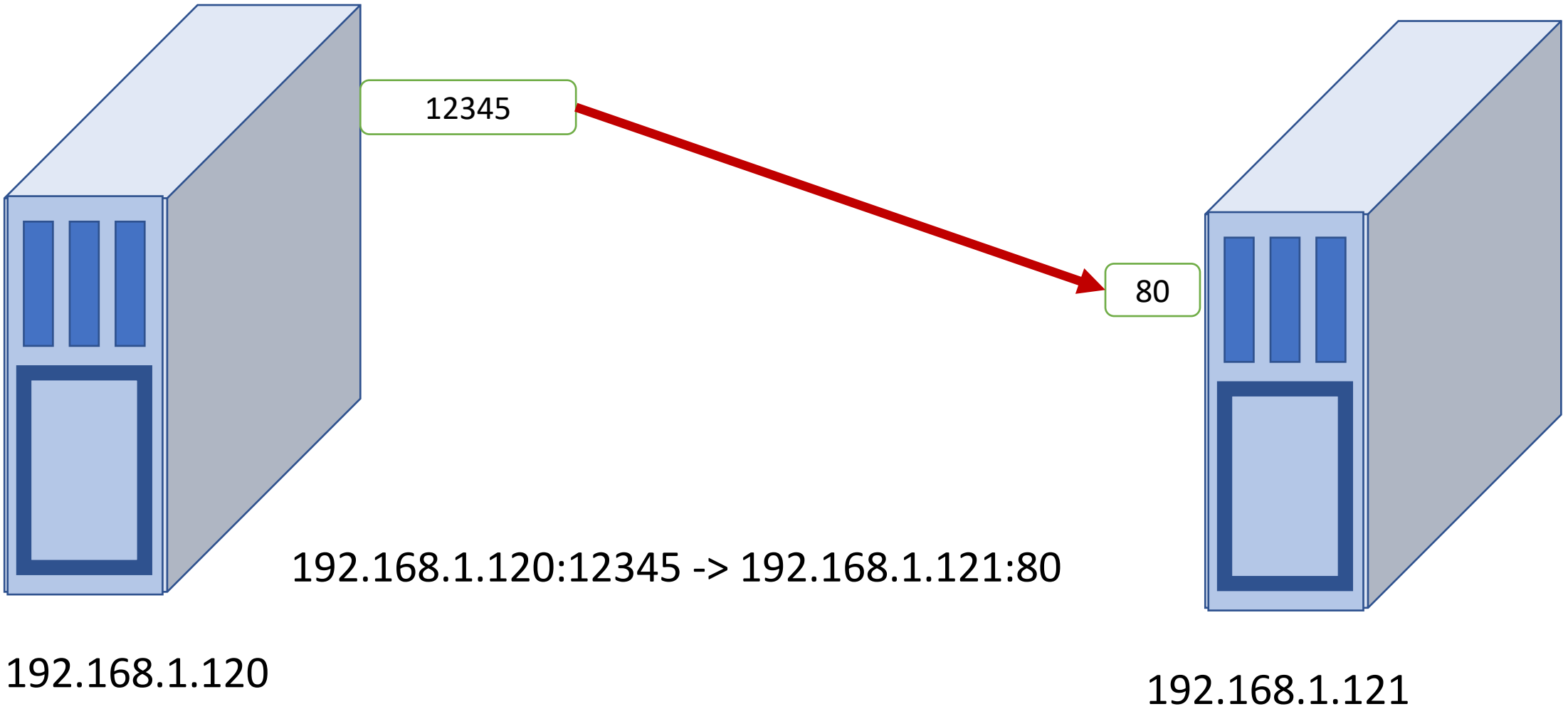

Internet
Localhost
127.20.0.xxx
GW
172.20.0.1

```
version: '3'
services:
  pinger:
    image: "pinger"
    ports:
      - "8893:8893"
    networks:
      - pingnet
    volumes:
      - ./data:/data
    environment:
      ServiceName: service_2
  pingrelay:
    build: "pingrelay"
    ports:
      - "8004:8894"
    networks:
      - pingnet
    volumes:
      - ./data:/data
    environment:
      ServiceName: service_1
networks:
   pingnet:

volumes:
  data: {}
```

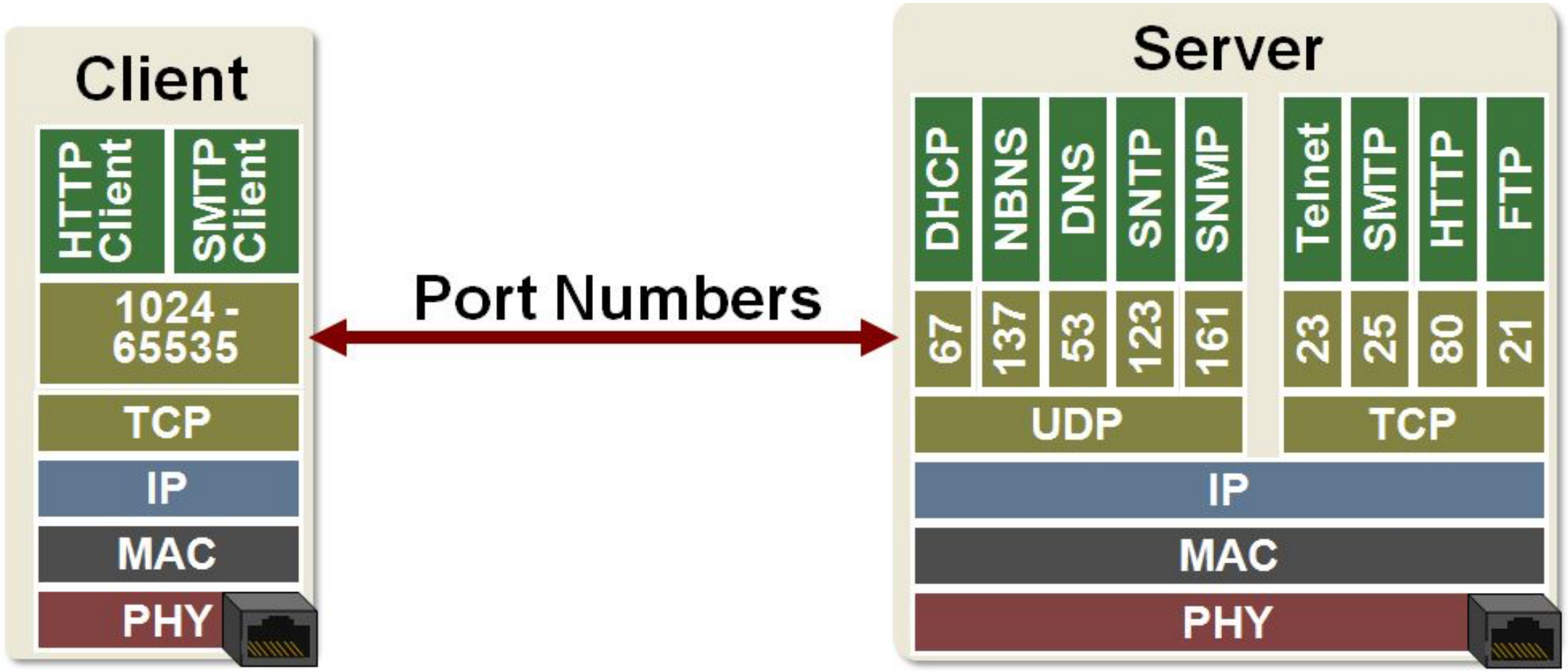


Do you see ”errors”?

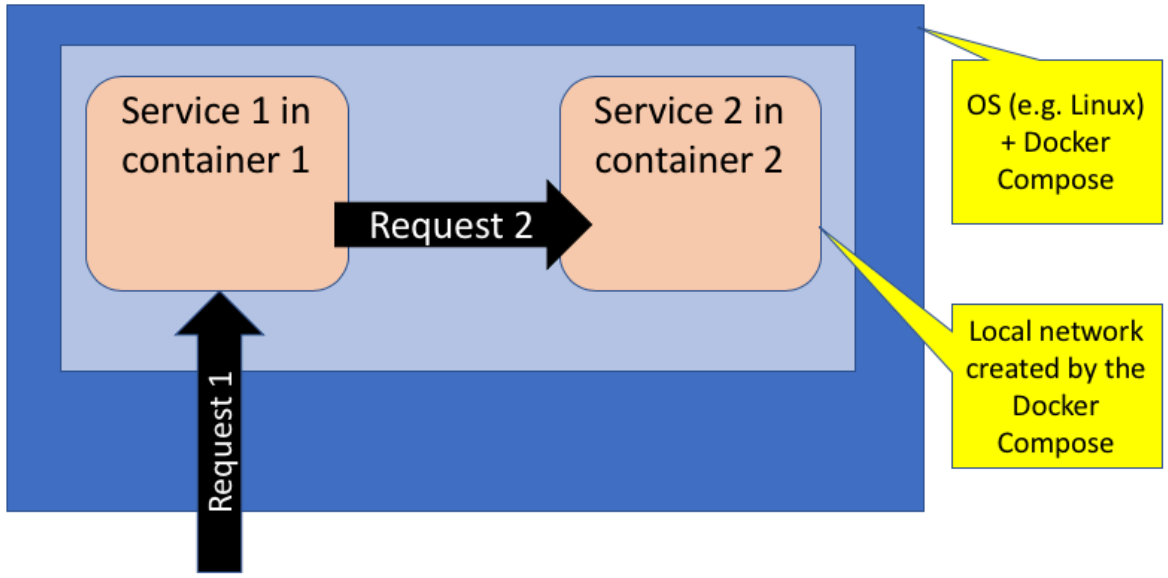
Something very basic



https://microchipdeveloper.com/tcpip:tcp-ip-ports



Your task



Service/application 1 should:

- As a response to incoming Request 1 send an HTTP GET request to Service2
- Compose a response from (4 lines of text)
  - “Hello from “ + <Remote IP address and port of the “ to “ + <Local IP address and port of Service1> Response of the above request to Service2
  - Return the composed response

Service/application 2 should

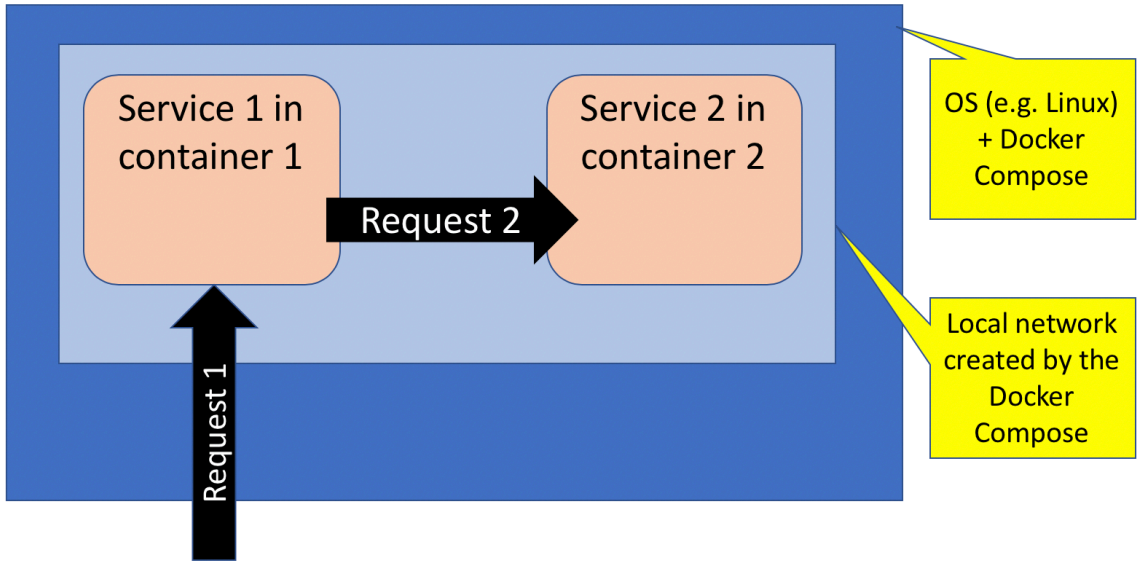
- As a response to incoming Request 2 compose a response from
  - “Hello from “ + <Remote IP address and port of the incoming Request2>
  - “ to “ + <Local IP address and port of Service2>
- Return the composed response

- By remote address/port we means the address of the host that sent the request. For example, in nodejs these can be tested with the following code:

```
http.createServer(function (req, res) {
 console.log("Req came from " + req.client.remoteAddress +
             ":" + req.client.remotePort);
 console.log("Req served at " + req.client.localAddress +
             ":" + req.client.localPort);
}).listen(port);
```

- Note that the above does not exactly meet requirements

Your task



- You should write *Dockerfiles* for the both services and *docker-compose.yaml* to start both containers so that Service1 is exposed in port number 8001. The docker-compose should also create a private network that allows Services 1 and 2 to communicate with each other but the only external Service 1.
- The service1 is assumed to be under development, so the image is rebuilt often (hint you may use ”build:” -primitive in *docker-compose.yaml*. Service2 is a reused service and you may pre-build the image. Image can be stored locally though.
- After the system is ready the student should return.
- Content of Docker and docker-compose.yaml files
- Explained response to Request 1 (that contains also response from Request 2). E.g. a Word or PDF-file where you also explain why the addresses and port-numbers are like they are. (We want to ensure that you understand how your program works).
- Source codes of the applications in some git.

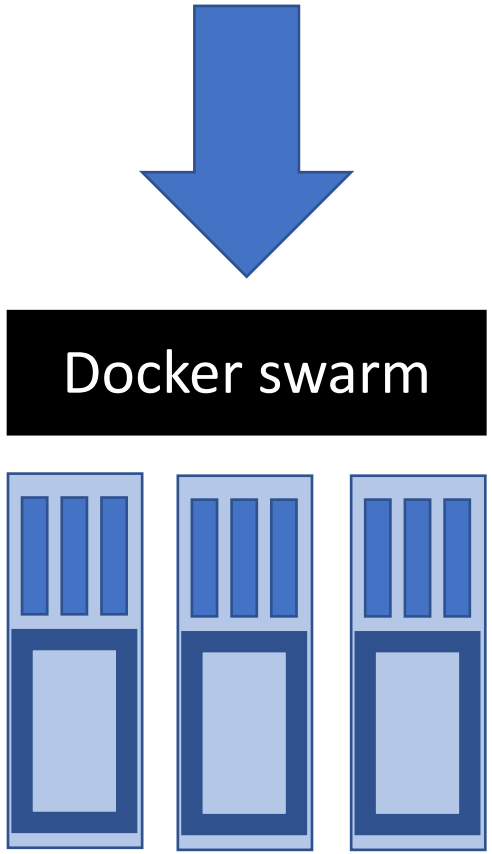
Tampereen yliopisto
Tampere University

How this will be checked

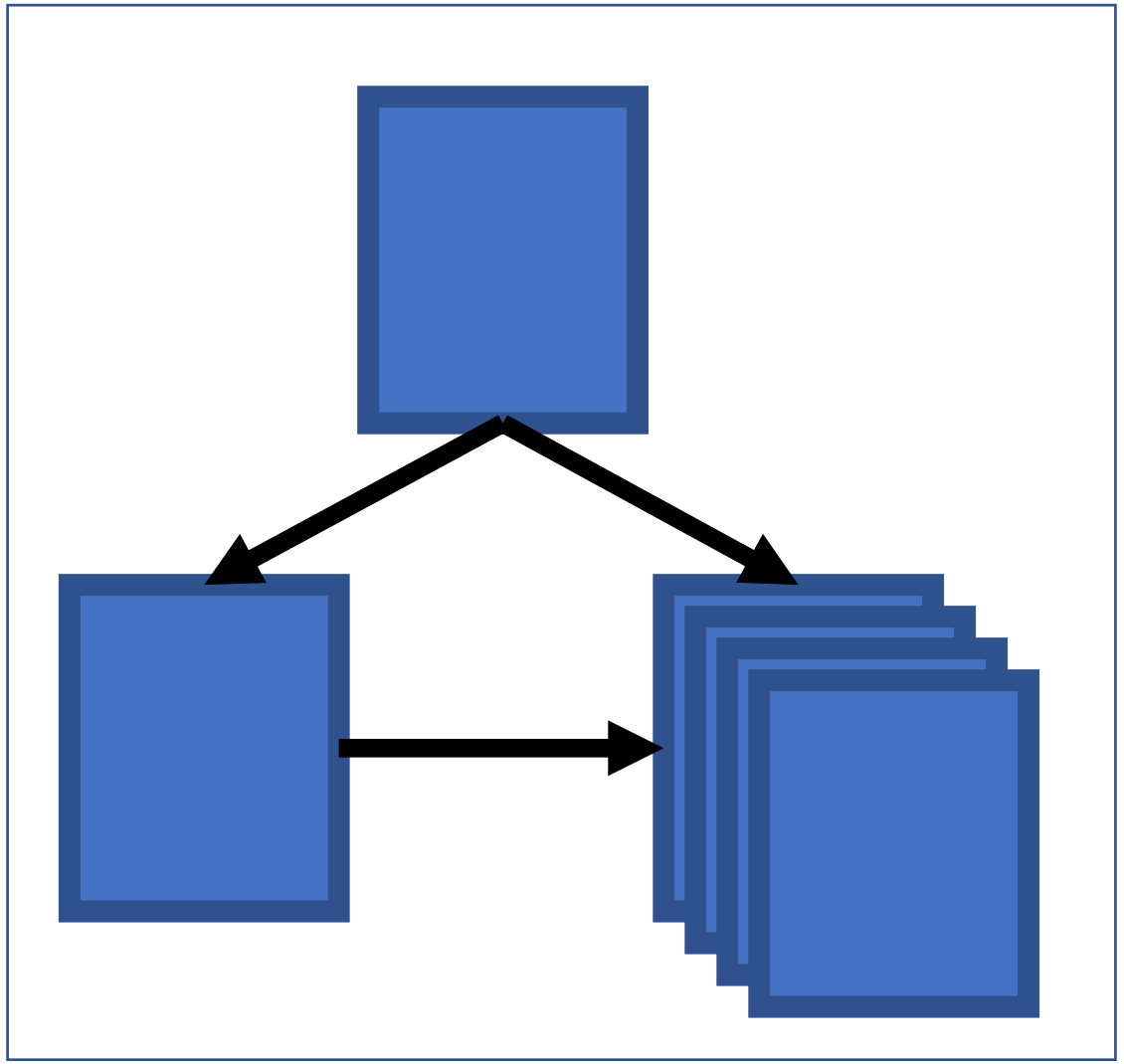
```
$ git clone <the git url you gave>
$ docker-compose up –build
$ curl localhost:8001
<output should follow the above requirement>
$ docker-compose down
```

Docker swarm - docker compose

Orchestration



Docket composes



Tampereen yliopisto
Tampere University

Hints

- Remember to backup your application and docker and compose files – you will need them in the future. E.g. to gitlab.
- It might be a good idea to create and test the applications first.
- You may need to visit https://docs.docker.com/compose/ and https://docs.docker.com/compose/networking/
- Docker images are easy to access, if they are tagged when build
- $ docker build --tag=pinger .
- If Docker image is rebuilt, docker-compose should also be given a hint that rebuilt should override the existing one
- $ docker-compose up --build

Tampereen yliopisto
Tampere University

Infrastructure as code

From: https://docs.microsoft.com/en-us/azure/devops/learn/what-is-infrastructure-as-code

Infrastructure as Code (IaC) is

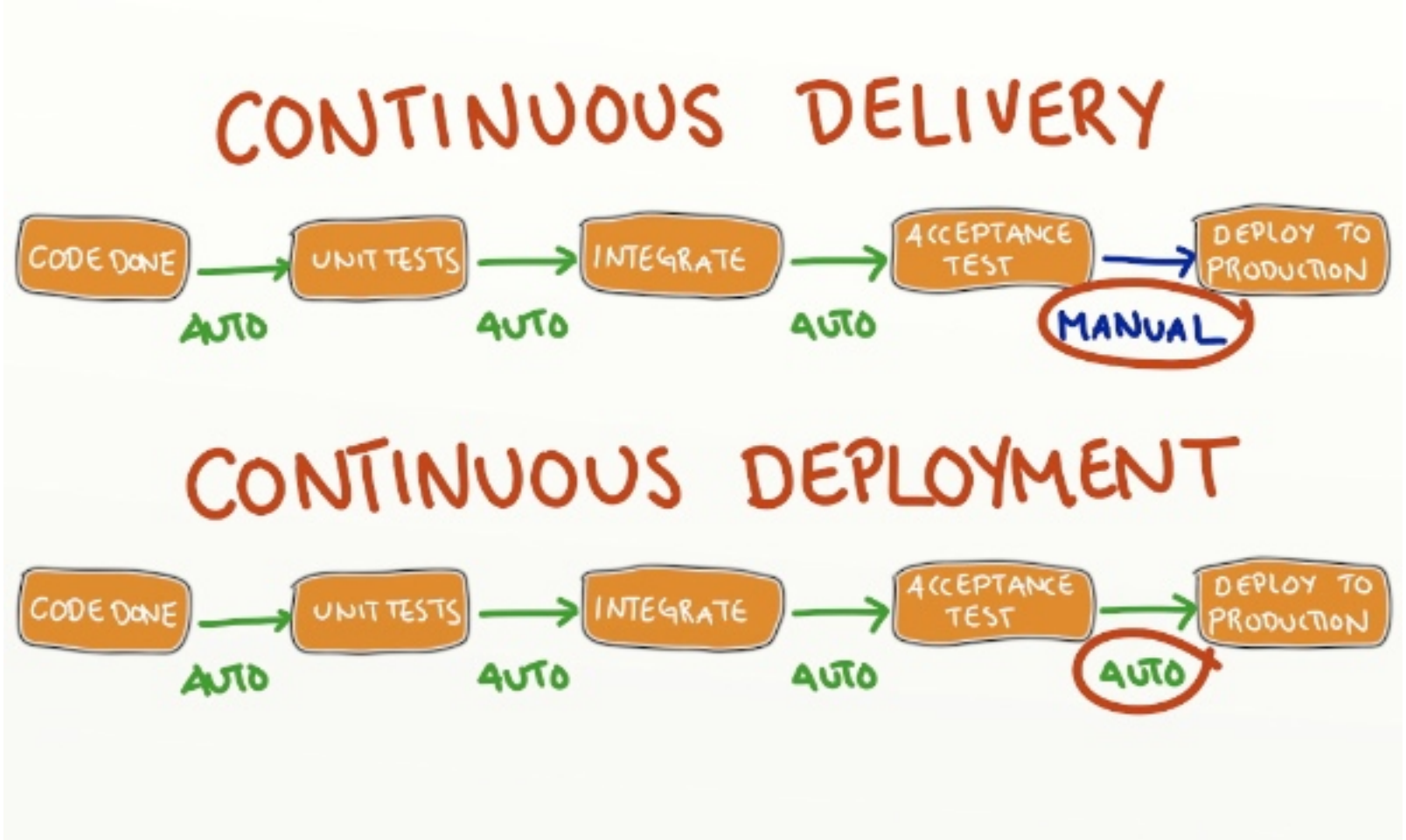
- the management of infrastructure (networks, virtual machines, load balancers, and connection topology) in a descriptive model,
- using the same versioning as DevOps team uses for source code.
- Like the principle that the same source code generates the same binary, an IaC model generates the same environment every time it is applied.
- IaC is a key DevOps practice and is used in conjunction with continuous delivery.

Tampereen yliopisto
Tampere University

Recap

Continuous delivery and deployment

(http://blog.crisp.se/2013/02/05/yassalsundman/continuous-delivery-vs-continuous-deployment)



Tampereen yliopisto
Tampere University

Perceived benefits

- **Improved delivery speed** of software changes Improved speed in the development and deployment of software changes to production environment.
- **Improved productivity in operations** work. Decreased communication problems, bureaucracy, waiting overhead due to removal of manual deployment hand-offs and organisational boundaries; Lowered human error in deployment due to automation and making explicit knowledge of operation-related tasks to software development
- **Improvements in quality**. Increased confidence in deployments and reduction of deployment risk and stress; Improved code quality; Improved product value to customer resulting from production feedback about users and usage.
- **Improvements in organisational-wide culture** and mind-set. Enrichment and wider dissemination of DevOps in the company through discussions and dedicated training groups ‘communities of practice’

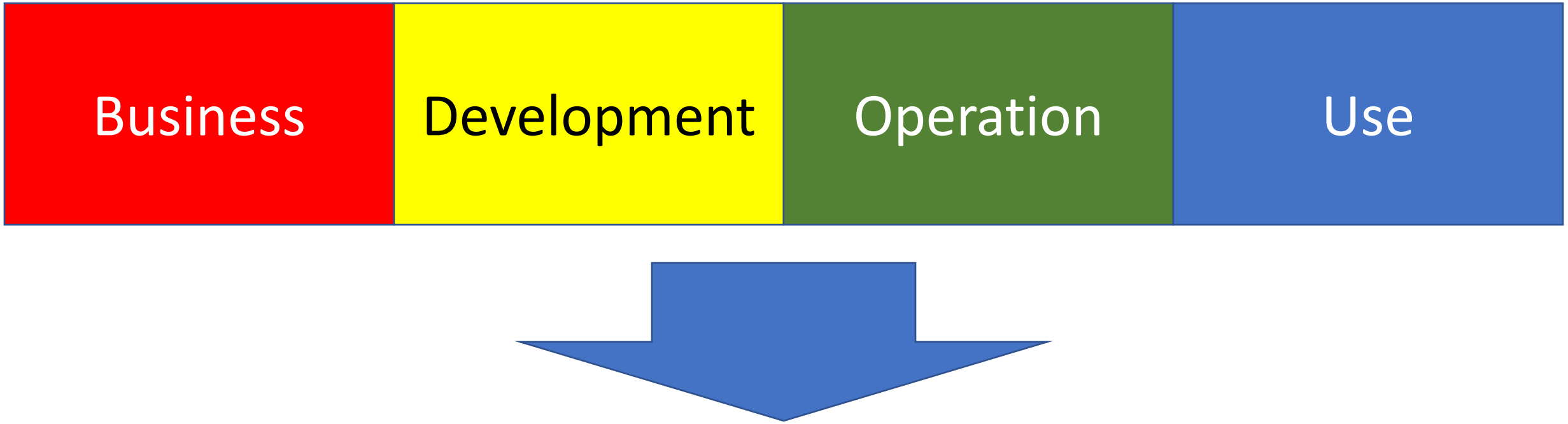
Tampereen yliopisto
Tampere University

Perceived challenges

- Insufficiencies in infrastructure automation
- High demand for skills and knowledge
- Project and resource constraints
- Difficulties in monitoring, especially for microservice-based applications and in determining useful metrics
- Difficulties in determining a right balance between the speed of new functionality and quality.

Tampereen yliopisto
Tampere University

Where was the beef?



Tampereen yliopisto
Tampere University

Safety critical systems (case medical systems)

T. Laukkarinen, K. Kuusinen and T. Mikkonen, "DevOps in Regulated Software Development: Case Medical Devices," *2017 IEEE/ACM 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track (ICSE-NIER)*, Buenos Aires, 2017, pp. 15-18, doi: 10.1109/ICSE-NIER.2017.20.

IEC 62304 - Clause 5.5.3	Software unit verification (verify before entering integration)	Continuous integration	Obstacle
IEC 62304 - Clause 5.8.5	Software development procedure (must be documented)	Development tools	Benefit
IEC 62304 - Clause 5.8.6	Software release (e.g. documents need to be ready)	Continuous deployment	Obstacle
IEC 62304 - Clause 5.8.6	Deployment repeatability	Deployment tools	Benefit
IEC 62304 - Clause 8	Item identification	Development tools	Benefit
IEC 82304-1 - Clause 8.4	Post-market reporting	Deployment tools	Benefit
IEC 82304-1 - Clause 8.4	Updating responsibility (customer should)	Continuous deployment	Obstacle

Embedded systems

- Very little research exists
- Lwakatare, L. E., Karvonen, T., Sauvola, T., Kuvaja, P., Bosch, J., Olsson, H. H., & Oivo, M. (2016). Towards DevOps in the Embedded Systems Domain: Why Is It So Hard? In 49th Hawaii International Conference on Systems Science, pp. 5437–5446. IEEE.

DevOps in the context of the embedded systems domain is challenging due to
- its dependency on hardware,
- limited visibility to customer environments,
- lack of deployment technology for customer-specific environments, and
- inability to get usage data from the customer environment to support the phenomenon

Tampereen yliopisto
Tampere University

Some examples of our research

Tampereen yliopisto
Tampere University

Need for speed – large national project http://n4s.dimecc.com/en/

- **Delivering Value in Real Time:** The Finnish software-intensive industry has renewed their existing business and organizations towards a value-driven and adaptive real-time business paradigm. Technical infrastructure and required capabilities have been established to support the transformation.
- **Deep Customer Insight—Better Business Hit-Rate:** Software-intensive industries in Finland are utilizing new technical infrastructure and capabilities as well as various sources of data and information to gain and apply deep insight into customer needs and behavior. This knowledge will enable the industry to improve sales and make significant returns on investment in the development of both products and services.
- **Mercury Business – Find the New Money:** This target focuses on how companies and societies can behave like liquid mercury, finding and flowing into new grooves. Mercury Business is the ability to adapt to new business conditions and search aggressively for business opportunities in new markets with minimum effort. This new approach to business growth is enabled by continuous and active strategic focus, a new leadership style.

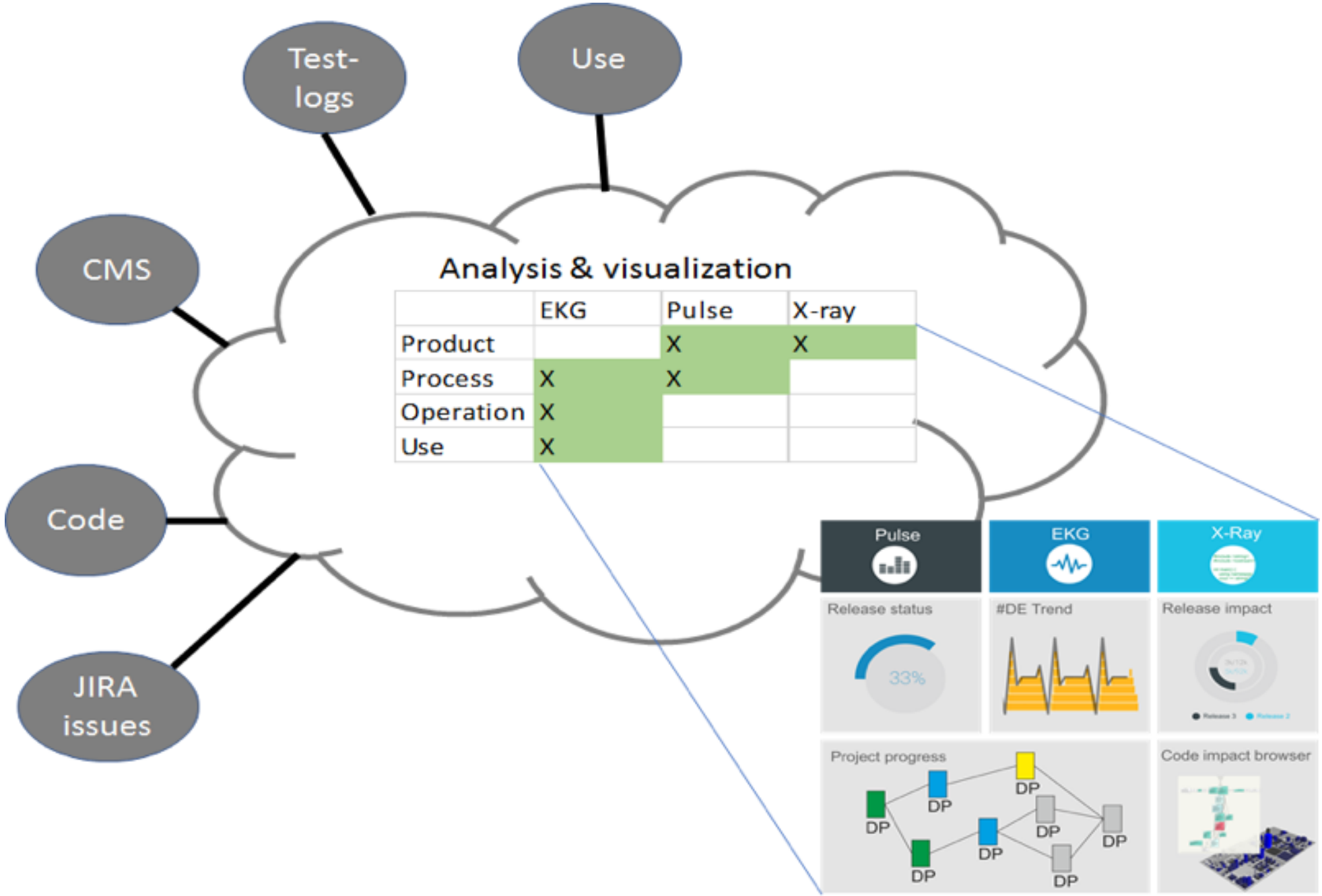
This project has also impacted our teaching.
(Hopefully positively ☺)

Tampereen yliopisto
Tampere University

VISDOM (iteavisdom.org)

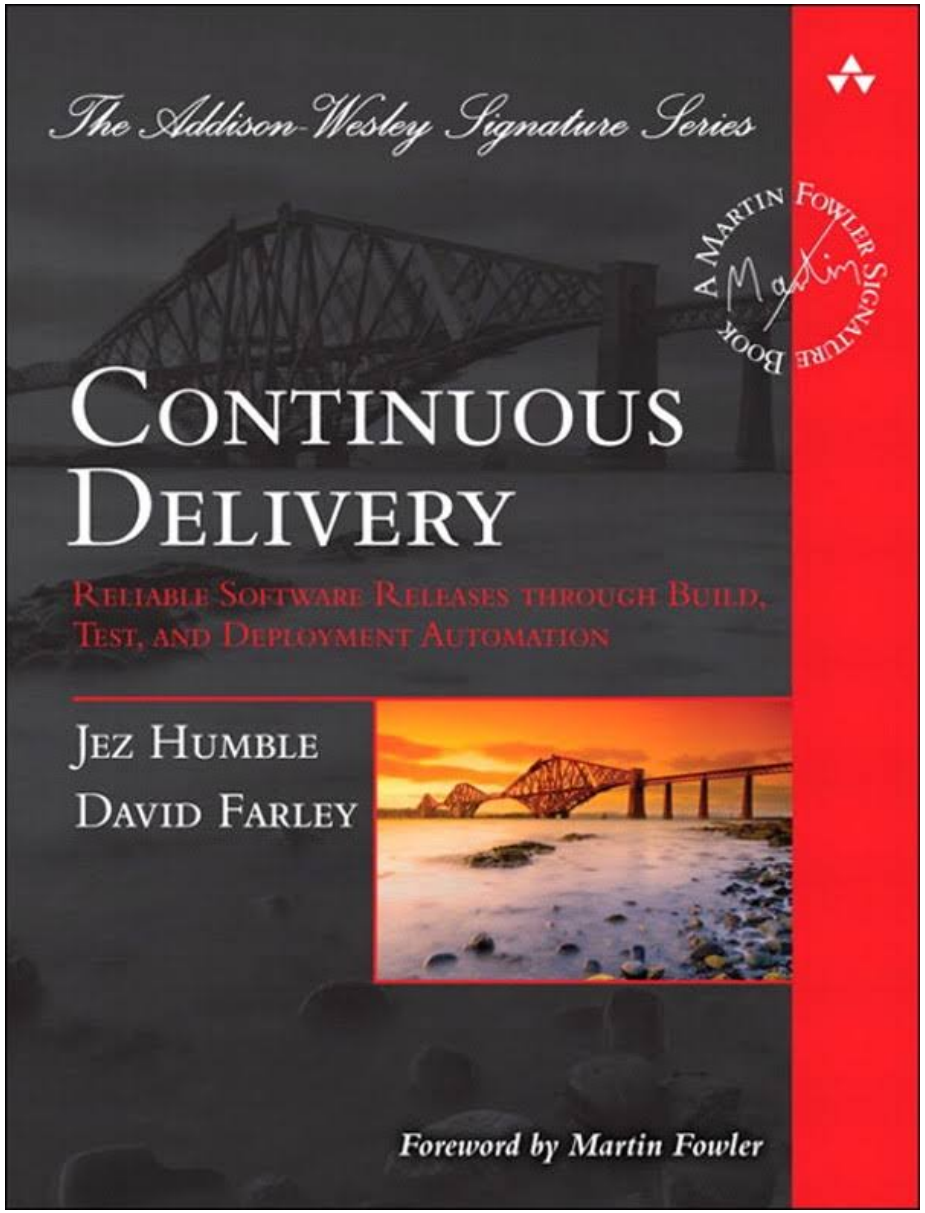
Visualisation is a powerful method for communication, especially in cross-disciplinary communication with various stakeholders, as in operations.

Many software development tools already provide some visualisations, but integrated views that combine data from several sources are still at research prototype level. The VISDOM project will develop new types of visualisations that utilise and merge data from several data sources in modern DevOps development. The aim is to provide simple “health check” visualisations about the state of the development process, software and use.


Test-logs
Use
CMS
Code
JIRA issues
Analysis & visualization
EKG
Pulse
X-ray
Product
X
X
Process
X
X
Operation
X
Use
X
Pulse
EKG
X-Ray
Release status
33%
#DE Trend
Release impact
Project progress
DP
DP
DP
DP
DP
DP
DP
DP
Code impact browser

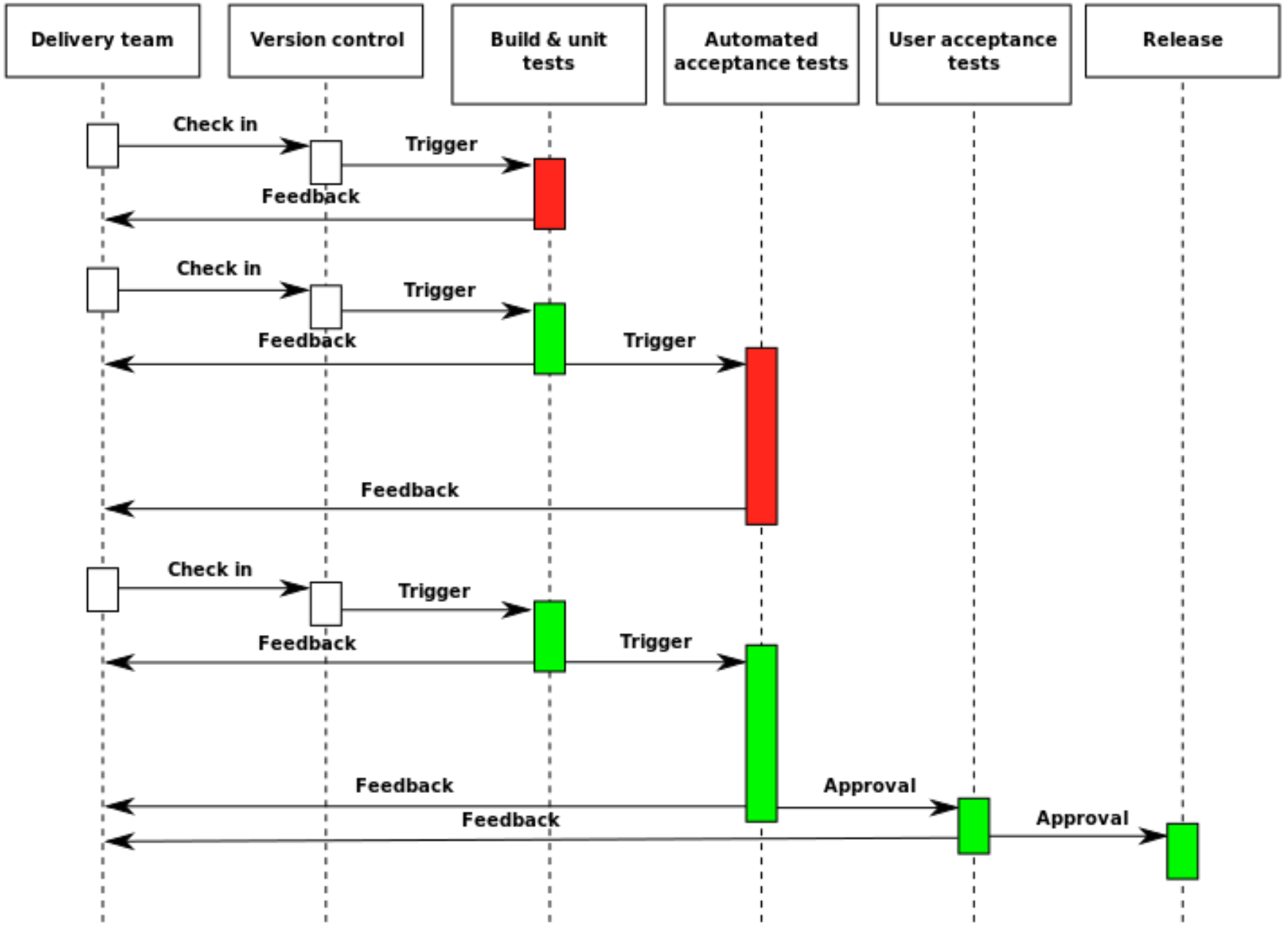
Tampereen yliopisto
Tampere University

CD: Some technical material



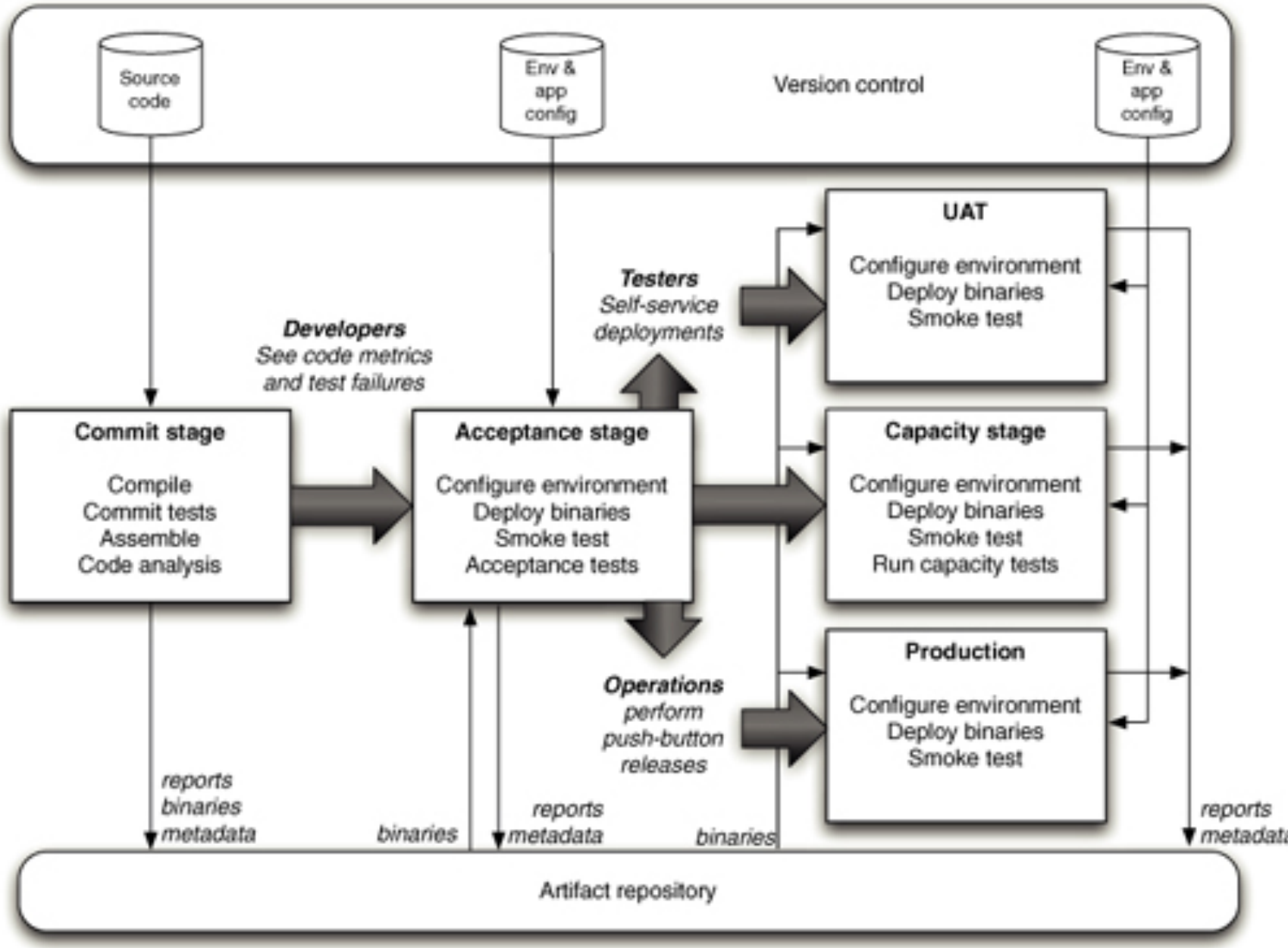
Tampereen yliopisto
Tampere University

Deployment pipeline (a possible example)

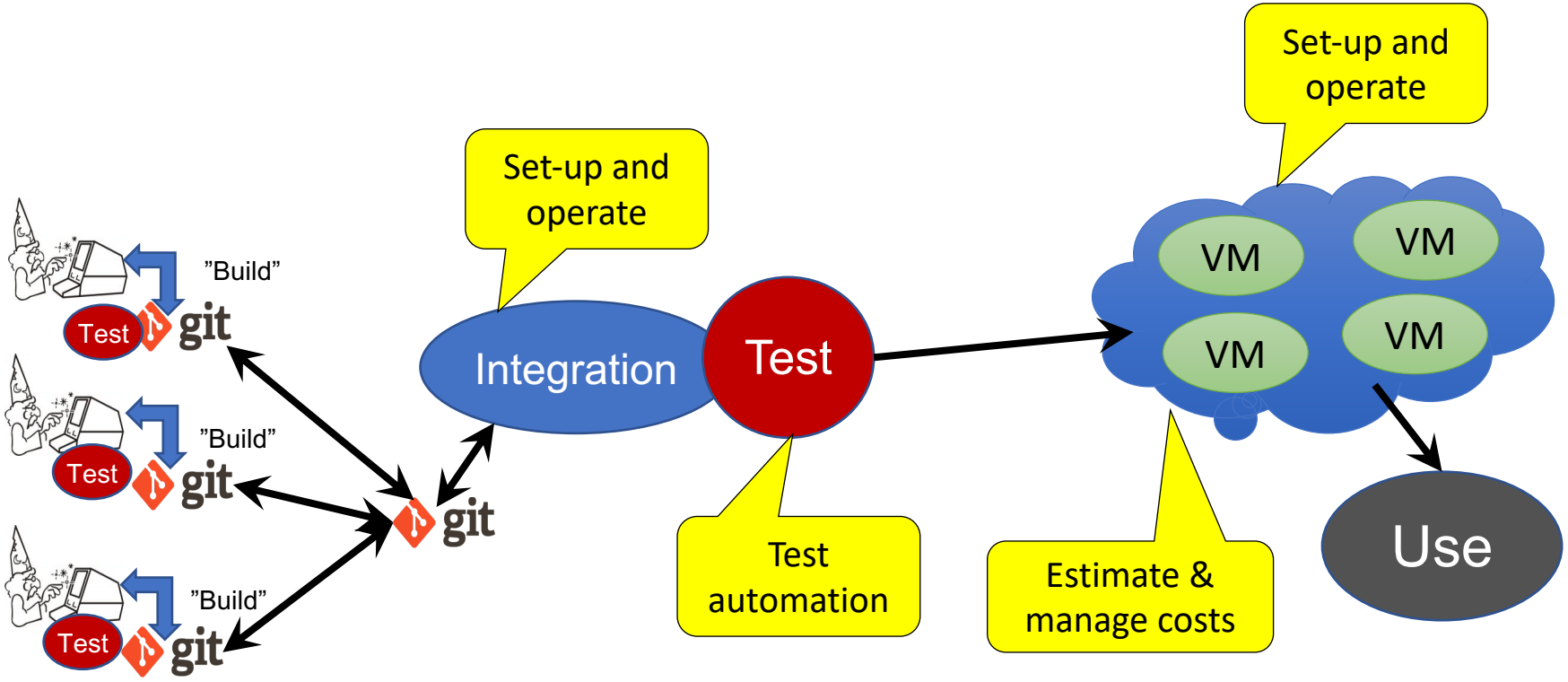


Tampereen yliopisto
Tampere University

Artifact repository



What does it really take to run CD?



Tampereen yliopisto
Tampere University

CI – essential practices
(according to Humbley and Farley)

- Don’t check in on a broken code
- Always run all commits tests locally before committing, or get your CI server to do it for you
- Wait for commit tests to pass before moving on
- Never go home on a broken build
- Always be prepared to revert to the previous revisions
- Time-box fixing before reverting
- Don’t comment out failing tests
- Take responsible for all breakages that result from your changes
- Test-driven development

Tampereen yliopisto
Tampere University

Deployment essential pract.
(according to Humbley and Farley)

- Only build your binaries once
- Deploy the same way to every environment
- Smoke-test your deployments
- Deploy to copy of production
- Each change should propagate through the pipeline instantly
- If any part of pipeline fails, stop the line